SYSTEM AND METHOD FOR EFFICIENT INPUT/OUTPUT OF A COMPUTER

SYSTEM

Attorney Docket Number

1292

5   Express Mail Label Number

EL528760141US

Inventor

Van Jacobson

Related Applications

10   This application claims the benefit of attorney docket

number 1197, application serial number 60/261,598, entitled

"Method and Apparatus for Efficient Input/Output of a

computer system filed on January 12, 2001 by Van Jacobson,

and is related to the subject matter of attorney docket

15   number 1196, application serial number 60/261,599 entitled,

"SYSTEM AND METHOD FOR OFFLOADING RESPONSES TO NETWORK

COMMUNICATIONS" filed on January 12, 2001 by Van Jacobson,

and each is incorporated herein by reference in its

entirety.

20   Field of the Invention

The present invention is related to computer hardware

and software and more specifically to computer hardware and

software for input/output interfaces.

Background of the Invention

Computer systems use conventional input/output subsystems to allow a computer system containing one or more processors to communicate with external systems.

5   However, there are several problems with conventional input/output subsystems.

One problem of conventional input/output subsystems is the use of interrupts. Because conventional input/output subsystems must communicate with external systems operating

10 independently of the computer system in which they operate, some conventional input/output subsystems employ interrupts to notify the processor of the computer system when an input has arrived or when an output has been sent. When a conventional input/output subsystem receives a

15 communication from the external system, the conventional input/output subsystem generates an interrupt to inform a processor that a communication has been received and is waiting to be serviced. If the processor provides output it may also receive an interrupt by the input/output system

20 to let the processor know that the output has been sent. Interrupts decrease the efficiency of a conventional computer system because the computer system must save its state to handle the interrupt, then restore the state after

the interrupt has been handled. Conventional computer systems used interrupts because they allowed a processor to perform other work in between handling the interrupts. Because the I/O used only a small portion of the

5 processor's throughput, interrupts offered a more efficient mechanism for processing I/O than conventional polling techniques. However, in modern multiprocessor systems in which the I/O consumes a more significant portion of processor resources, the benefits of using interrupts can

10 be outweighed by the overhead required.

Another problem with conventional I/O subsystems is the lack of speed of conventional I/O busses. I/O bus speeds in conventional computer systems have not kept pace with the increases in modern processor speeds, and thus,

15 many computer systems are constrained by the speed of the I/O bus. If the computer system is operating as a server, such as a web server or a file server, the bottleneck at the I/O bus may leave the processors idle while the I/O is being sent or received.

20 To avoid the bottleneck of the I/O subsystem, some computer systems have employed direct memory access techniques. Direct memory access allows the I/O subsystem to access the same memory facilities being used by the one

3

or more processors of the computer system. The processor
or processors place the data to be transferred into the
memory, and instruct the DMA subsystem to transfer the data
to an external system. Data received by an external system

5 is sent by the DMA subsystem to the memory. Interrupts are
used to inform the processor that the DMA subsystem has
received data from the external system and also to inform
the processor that the data has been sent to the external
system. Although the use of DMA reduces the number of

10 interrupts, interrupts are still used with DMA subsystems.
Furthermore, data transfers are made from the external
system to the DMA subsystem using the I/O bus. In
addition, the memory bus throughput is reduced because all
data transfers require two uses of the memory bus: once to

15 put the data into memory and once to remove it.
Furthermore, DMA circuitry requires memory data bus
contention control systems because both the processor and
the DMA subsystem can access the memory data bus at the
same time.

20   Another problem associated with using the memory
facilities of a computer system for I/O processing is that
the responses to the I/O requests in modern server systems
could simply overwhelm any one memory bus. While many I/O
requests such as disk I/O requests or requests for web

pages are relatively small, the response to the request is
often large compared to the request: it may contain an
entire file or a web page. Using the memory facilities of
the computer system to process I/O could overwhelm the
5 memory bus while the response to the request is being
transmitted.

Because many computer systems utilize several
processors, and because certain requests for information
may be distributed across several messages, it can be most
10 efficient to provide all of the messages in a request to
the same processor that handled any prior message in a
request. Some I/O subsystems employ complicated
interprocess communication techniques in order to route
such messages to the proper processor. The overhead
15 required for these techniques can reduce the throughput of
the processors handling the messages.

What is needed is a system and method that has a data
rate exceeding conventional I/O bus speeds, does not employ
interrupts, will not overwhelm the memory facilities of a
20 computer system in which it is employed, and can
efficiently route messages to an appropriate processor in a
multiple processor computer system.

Summary of Invention

A system and method accepts communications directly
into the addressable main memory of the computer system,
bypassing the memory bus, while allowing retrieval at

5 memory speeds. A pointer to the communication in memory is
queued to allow processors to retrieve the communication
without the use of interrupts. Each communication may be
queued for a specific processor or other entity, such as
the processor that handled the last communication from the

10 same source and to the same destination that had the same
protocol as the communication. When the processor or other
entity processes the communication, it may provide a
response or other communication into the addressable memory
of the computer system. The system and method retrieves

15 the response or other communication directly from the
memory and provides it as output, for example, to a
network. If multiple processors have access to shared
memory and also have access to their own private memory,
the first communication may be accepted into shared memory

20 and the response or other communication may be stored into
private memory for the processor processing he
communication, spreading he various communications across
different memory buses. In the case in which the first
communication is relatively small, such as a request for

information, and the response or other communication is large, such as the information requested being a web page or a file from a storage device, processing all of the requests via a shared memory bus and the responses via

5   private memory busses of multiple processors exploits the asymmetry of the communications, to provide a throughput larger than would be possible using a single bus without overwhelming any of the memory buses.

Brief Description of the Drawings

10   Figure **1A** is a block schematic diagram of a conventional computer system.

Figure **1B** is a block schematic diagram of a conventional computer system.

Figure **2A** is a block schematic diagram of a system for

15   communicating with systems external to a computer system according to one embodiment of the present invention.

Figure **2B** is a block schematic diagram of an incoming dispatch storage **214** of Figure **2A** according to one embodiment of the present invention.

20   Figure **2C** is a block schematic diagram of a table of entries stored in a dispatch storage **288** of Figure **2B** according to one embodiment of the present invention.

Figure **2D** is a block schematic diagram of an incoming dispatch storage **214** of Figure **2A** according to another embodiment of the present invention.

Figure **3** is a flowchart illustrating a method of

5   processing input/output of a computer system according to one embodiment of the present invention.

<u>Detailed Description of a Preferred Embodiment</u>

A.   <u>**Computer Architectures**</u>.

The present invention may be implemented at least in

10   part as computer software on a conventional computer system. Referring now to Figure **1**, a conventional computer system **150** for practicing the present invention is shown. Processor **160** retrieves and executes software instructions stored in storage **162** such as memory, which may be Random

15   Access Memory (RAM) and may control other components to perform the present invention. Storage **162** may be used to store program instructions or data or both. Storage **164**, such as a computer disk drive or other nonvolatile storage, may provide storage of data or program instructions. In

20   one embodiment, storage **164** provides longer term storage of instructions and data, with storage **162** providing storage for data or instructions that may only be required for a shorter time than that of storage **164**. Input device **166**

8

such as a computer keyboard or mouse or both allows user
input to the system **150**.   Output **168**, such as a display or
printer, allows the system to provide information such as
instructions, data or other information to the user of the
5   system **150**.   Storage input device **170** such as a
conventional floppy disk drive or CD-ROM drive accepts via
input **172** computer program products **174** such as a
conventional floppy disk or CD-ROM or other nonvolatile
storage media that may be used to transport computer
10   instructions or data to the system **150**.   Computer program
product **174** has encoded thereon computer readable program
code devices **176**, such as magnetic charges in the case of a
floppy disk or optical encodings in the case of a CD-ROM
which are encoded as program instructions, data or both to
15   configure the computer system **150** to operate as described
below.

In one embodiment, each computer system **150** is a
conventional Sun Microsystems Ultra 10 workstation running
the Solaris operating system commercially available from
20   Sun Microsystems of Mountain View, California, a Pentium-
compatible personal computer system such as are available
from Dell Computer Corporation of Round Rock, Texas running
the Windows (95, 98 or NT) operating system commercially
available from Microsoft Corporation of Redmond Washington

9

or a Macintosh computer system running the MacOS

commercially available from Apple Computer Corporation of

Cupertino, California and the Netscape browser commercially

available from Netscape Computer Corporation of Mountain

5   View, California although other systems may be used.

Referring now to Figure **1B**, another aspect of an

architecture of a conventional computer system **150** is

described according to one embodiment of the present

invention. CPU A **182**, CPU B **184** and CPU C **186** are

10   conventional microprocessors such as the conventional

Ultrasparc II, Ultrasparc III or MAJC microprocessor

commercially available from Sun Microsystems, Inc. of Palo

Alto, California. Each of the CPUs **182-186** is coupled to a

conventional memory bus **188** such as the conventional Front

15   Side Bus specified by Intel Corporation of Santa Clara,

California. Each of the CPU's has access to two types of

memory via the front side bus **188**. Shared memory **192** is

available to all CPUs **182-186** via a commonly shared main

memory bus. Private memory A **198** is available to CPU A **182**

20   via a separate main memory bus, but not CPU B **184** and CPU C

186. Private memory B **194** is similarly available to CPU B

184 via a separate main memory bus, but not the other CPUs

182, 186. Private memory C **196** is similarly available to

CPU C **186** via a separate main memory bus, but not the other

CPUs **182, 184.** I/O bus **190** is available to the CPUs **182-186** and coupled to interrupt driven I/O interfaces **189, 191** such as conventional ISA or PCI interfaces.

In one embodiment, the systems that make up elements
5 **189** and **191-198** may each be made of any number of physical devices such as memory or interface cards, or may be combined onto a smaller number of such physical systems. In one embodiment, I/O interface B **191** may include one or more communication interfaces such as conventional TCP/IP-
10 compatible communication interface cards that are capable of communicating with one or more external devices **178** via a conventional network **176** such as an Ethernet or the Internet. Devices **178** may be conventional computer systems or disk drives or any other device that may wish to
15 communicate with computer system **150**.

The present invention may be used with any system architecture and not just the architecture shown in Figure **1B**, however that architecture will be used for purposes of example. As an overview of the system, the present
20 invention can include one or more cards that have two sets of interfaces. The first set of interfaces are memory interfaces which appear to CPUs **182-186** as if they are conventional memory devices **192-198**. The present invention

may be used in addition to conventional memory systems **192-198**. The second set of interfaces are interfaces to devices **178**, such interfaces acting like conventional I/O interfaces.

5   **B. Receiving Input.**

Referring now to Figure **2A**, a system for communicating with systems external to a computer system is shown according to one embodiment of the present invention. Incoming communication interface **210** is a conventional

10   communication interface for communicating with devices coupled to input **208**, such devices being external to the computer system containing incoming communication interface **210**. In one embodiment, input **208** is coupled to the output of a device that intercepts and responds to syn packets and

15   forwards other packets to its output (coupled to input **208**) as described in the copending application, described above.

Incoming communication interface **210** may include a conventional TCP/IP-compatible communication interface, a conventional Ethernet interface, any other form of

20   communication interface capable of communicating with a storage device such as a disk drive or tape drive or any other form of interface. Incoming communication interface

12

210 passes a portion or all of each such communication received to incoming interface manager **212**.

Incoming interface manager **212** receives incoming communications and stores a portion or all of each of them into incoming/dispatch storage **214** as described in more detail below. As described in more detail below, in one embodiment, incoming/dispatch storage **214** contains conventional storage such as memory or disk storage which may be arranged as a circular buffer, with pointers to the head and tail of the buffer.

Incoming communication interface **210** thus stores the some or all of the communication into incoming/dispatch storage **214** "directly". As used herein, to store something "directly" means to store it into a storage location accessible to at least one processor via a main memory bus of the computer system, but where such storage is performed without the use of any system I/O bus.

In one embodiment, incoming/dispatch storage **214** contains conventional 2 port or multiport RAMs such as those commercially available from Cypress Semiconductor of San Jose, California: one port is coupled to incoming communication interface **210** and the other port is coupled to the main shared memory bus of the computer system. This

13

allows incoming communication interface **210** to store some

or all of the communication "sinebusly", that is, without

the use of a system memory bus, such as the main memory bus

used by processors **230-234**.

5   The storage is also performed "sineinterruptusly." As

used herein, "sineinterruptusly" means without causing an

interrupt to be generated to a main processor of the

computer system. A main processor of the computer system

is a processor that is under control of the operating

10   system of the computer system, as opposed to processors

dedicated to I/O functions which may reside on interface

cards or peripherals and operate somewhat independently of

the operating system.

Shared memory interface **218** is a conventional memory

15   interface that may be coupled to a conventional memory bus

such as a front side bus interface and allows shared memory

access in one embodiment, or private memory access in

another embodiment. In one embodiment, shared memory

interface **218** includes a capability to allow one or more

20   system processors to read one communication of the contents

of a buffer in incoming/dispatch storage **214** at the

location pointed to by a pointer to one of the FIFOs in

incoming/dispatch storage **214** described in more detail

below, and then updates the head pointer to point to the memory location immediately following the communication in incoming/dispatch storage **214**. In another embodiment, shared memory interface **218** allows one or more system

5 processors to read the pointers or the contents of the buffer in incoming/dispatch storage **214** so that, for example, the device can handle the details of updating the head pointer.

Referring now to Figures **2A** and **2B**, incoming dispatch

10 storage **214** of Figure **2A** is shown in more detail in Figure **2B** according to one embodiment of the present invention. Input/output **290** may be coupled to both incoming interface manager 212 and shared memory interface **218**. The remaining elements **280-288** may simply be portions of the storage of

15 incoming/dispatch storage **214** that have a specific function as described below. A portion of incoming/dispatch storage **214** is set aside for use as the conventional double circular buffer into which some or all of each communication is stored as described above. The address of

20 the head of the buffer is stored in head/tail pointer storage **284** along with the tail, which are used as will now be described. When a communication is received by incoming interface manager **212**, it stores some or all of the communication as a block into the buffer beginning with the

15

location specified by the tail pointer as described above. The tail pointer may be an address or an offset or any other identifier that could be used to calculate a location of buffer **280**. Incoming interface manager **212** then updates

5   the tail pointer to point to the address after the last byte stored in buffer **280**.

In the embodiment of Figure **2B**, the buffer **280** is treated as having a circular arrangement with the lowest numbered byte considered to immediately follow the highest

10 numbered byte in the buffer **280**. When information interface manager **212** reaches the end of the buffer **280**, it continues storing at the beginning of the buffer **280**. If the last byte for a communication stored is at the end of the buffer, the next storage location in the buffer will be

15 the lowest numbered address in the buffer **280** and so incoming interface manager **212** stores that address as the new tail in head/tail pointer storage.

Information is read from the head of the buffer **280** as pointed to by a head pointer in head/tail pointer storage

20 **284**. The device reading the information from the head of the buffer adjusts the head pointer to the address in buffer **280** corresponding to start of the next block of data to be read. In one embodiment, the size of each block is

16

stored in the first one or more bytes of the block by

incoming interface manager **212** and used to read the next

block and adjust the head pointer. In another embodiment,

each block of data has a fixed size, so no size is stored

5 with the block.

As described herein in one embodiment, one of the

processors **230-234** reads blocks of data and routes them

among the other processors **230-234** excluding or including

the processor performing the routing function. For

10 purposes of example, processor A **230** will be described as

performing this function, and processor A **230** routes the

blocks to processors B and C **232, 234**. In another

embodiment, the routing function described below may be

performed by incoming interface manager **212** as it stores

15 the blocks into buffer **280**. In still another embodiment, a

single processor is used or used to process the blocks

received and so no routing function is performed.

Processors **230-234** may be the same as CPUs **182-186** of

Figure **1B** or may be different from those CPUs.

20   Processor A **230** reads the communication stored

beginning at the pointer to the head of the buffer **280**,

such head pointer being stored in head/tail pointer storage

284. This may be performed by reading the head pointer in

17

head/tail pointer storage **284**, internally storing a copy of the head pointer, reading from buffer **280** the block indicated by the head pointer and updating the head pointer in head/tail pointer storage **284** to point to the next block

5 in one embodiment as described above. In another embodiment, shared memory interface **218** handles the details of maintaining buffer **280** and head pointers and reading at the proper location of buffer **280**: processor A simply reads from a specific block and the next block in buffer **280** is

10 provided by shared memory interface **218**, which updates the head in head/tail pointer storage **284**.

Processor A **230** assigns the block to a processor **232-234** that will process the communication. In one embodiment, as part of this assignment, processor A **230**

15 compares some or all of the block to entries in dispatch storage **288** and if a corresponding entry is located, processor A **230** assigns the block to a processor **232-234** indicated in the entry in dispatch storage that corresponds to the block as described in more detail below.

20 For example, if the blocks in buffer **280** are TCP packets, each block will contain a TCP header of the packet. The TCP header may be used to determine which processor **232-234** should process the packet. When a packet

18

is assigned to a processor **232-234** as described in more detail below, the source IP address, source port, destination IP address, destination port and protocol from the block may be copied by processor A **230** from the TCP

5   header portion of the block into an entry in a table of entries stored in dispatch storage **288** of Figure **2B** along with an identifier of the processor to which the packet is assigned as described below. A representative table of entries is shown in Figure **2C**. When processor A **230**

10   retrieves another block from buffer **280**, processor A **280** compares the source IP address, port, destination IP address, destination port and protocol from the packet header of the block read from buffer **280** with the entries in the table in dispatch storage **288** and if a matching

15   entry is located, the communication is assigned to the processor **232-234** listed in the matching entry in dispatch storage **288**.

If no matching entry is found, the block may be assigned to a processor **232-234** by random selection, round-

20   robin selection, in which blocks are assigned to processors using a repeating order of the processors, or may be assigned to the processor **232-234** that has the shortest queue of pending communications waiting to be processed (as

described in more detail below) or may be assigned using other forms of load balancing.

The block may be assigned to a processor using other criteria. For example, in another embodiment, only the destination IP address of the packet header in the block is used to assign the packets to a processor **232-234**. For example, if processor B **232** handles communications to one IP address and processor C handles **234** communications to two other different IP addresses, the entries in dispatch storage **288** may include the destination address and processor identifier and may be preassigned, and not dependent on an earlier assignment. Processor A **230** will match the destination IP address with an entry in the table and assign the packet to the processor having the matching entry, but need not add an entry to the table in dispatch storage **288**. In one embodiment, two tables are used. A first one relating destination IP addresses to processors as described above in this paragraph, and a second one relating source and destination IP addresses and ports as well as protocols to processors as described earlier. If the destination address from a block being assigned is not located in the first table, the second table is used as described above to identify a processor to which the block

should be assigned or a new entry is added into the second table as described above.

To complete the assignment of a communication to a processor **232-234**, processor A **230** places the pointer it copied from head/tail pointer storage **284** into processor B FIFO **286** or processor C FIFO **288** depending on whether the assignment was made to processor B **232** or processor C **234**, respectively. The block itself may remain in buffer **280**, although in another embodiment, processor A **230** moves the block itself into Processor B FIFO **286** or processor C FIFO **288** instead of the pointer or moves the block to a different buffer (not shown) that may be shared or dedicated to a processor and inserts into FIFO **286** or **288** a pointer to the block in this buffer. In the embodiment in which a buffer is not dedicated to the processor, processor B FIFO **286** and processor C FIFO **288** are arranged as conventional double buffers with a head and a tail pointer (in a reserved location of each respective FIFO **286, 288** not used to write pointers or blocks as shown in t he Figure) pointing to the head and tail of the FIFO and processor A **230** moves either the appropriate pointer or the block to the tail of the FIFO **286, 288** and updates the tail to the next available location in the FIFO. If a buffer is dedicated to each processor is used, head and tail pointers

21

are used to point to the head and tail of the buffer itself, with the buffer acting as the FIFO in a double buffer arrangement as described above.

In the embodiment in which processor A **230** assigns the
5   block to the processor having the shortest queue, processor A **230** may determine which queue is shortest by comparing the difference between the head and the tail of each FIFO **286, 288** and choosing the processor **232-234** corresponding to the FIFO having the smallest difference.

10 As described above, one incoming communication interface and one incoming interface manager feeds one incoming dispatch storage and is coupled to the memory bus via shared memory interface **218**. However, any number of these in any ration to one another may be used in other
15 embodiments of the present invention.

C.   Processing.

Processors **232-234** monitor their respective FIFOs and when the head and tail are not pointing to the same position, retrieve either the block at the head of the FIFO
20 **286, 288** or the block in buffer **280** pointed to by the head of the respective FIFO **286, 288**, (or in another FIFO as described above) and then advance the head of the respective FIFO **286, 288** past the end of the pointer or

22

block stored in the respective FIFO **286, 288**. In another

embodiment, each processor **232, 234** reads from a fixed

location of shared memory interface **218** and receives any

block corresponding to the pointer at the head of the FIFO

5 **286, 288** corresponding to the location from which the block

was read. If the block is not empty, processor **232, 234**

processes the block as described below. In such

embodiment, shared memory interface **218** supplies the block

from the buffer **280** and manages the head and tail of the

10 FIFOs **286, 288**.

Each processor **232** or **234** then processes the block it

retrieves and may optionally provide a response to the

block. If the block is a request for a web page, processor

232 or **234** retrieves or generates the web page and provides

15 it as described in more detail below. If the communication

is a request for some or all of a file stored on a drive,

processor **232** or **234** retrieves and provides some or all of

the file.

The assignment and retrieval process described above

20 can operate entirely as a memory operation via the shared

memory of the computer system. Because each of the

processors **230-234** have access to the shared memory, no

complex interprocessor communication is required.

23

In one embodiment, incoming dispatch storage **214** has the arrangement shown in Figure **2D**. In such embodiment, instead of head/tail pointer storage **284**, available locations FIFO **292** is employed. Buffer **280** need not be arranged as a circular buffer. Instead, buffer **280** contains fixed-length blocks, and pointers to each such block are initialized into available locations FIFO **292**, which is arranged as a circular buffer with pointers to the head and tail of the FIFO **292**. (All initializations of incoming dispatch storage **214** may be performed by incoming interface manager **212** on power on or reset.) When incoming interface manager **212** receives a communication, it extracts the portion to be saved as a block and retrieves from available locations FIFO **292** the pointer to buffer **280** pointed to by the head of available locations FIFO **292**, stores the block at the location corresponding to the pointer removed, and advances the head of available locations FIFO **292** to point to the next location in available locations FIFO **292**. If incoming interface manager **218** routes the block as described above, the pointer removed is placed at the location of processor B FIFO **286** or processor C FIFO **288** indicated by the respective FIFO's tail corresponding to the processor **232**, **234** to which the block was assigned.

24

If one of the processors **230-234** assigns the block to

a processor, an additional FIFO (not shown) may be used

that is arranged as FIFOs **286** and **288** described above, and

incoming interface manager **212** places the pointer it

5   retrieves from available locations FIFO **292** onto the

additional FIFO at the location indicated by its tail and

advances the tail. Processor A **230** assigns blocks by

retrieving them from the head of the additional FIFO,

advancing its head, and then adding the pointer to the tail

10   of the FIFO **286, 288** corresponding to the processor to

which the block was assigned as described above.

When a processor **232, 234** retrieves for processing the

block pointed to by the pointer to the buffer **280** at the

head of its respective FIFO **286, 288** it places the pointer

15   at the location pointed to by the tail of available

locations FIFO **292** and advances the tail to the next

location in the FIFO, signaling that the block in buffer

280 may now be reused.

D.   **Output**.

20   To generate output to an external system, processor B

232 and processor C **234** write any response to the block to

their respective private memory interface **250** or **260**, each

providing a conventional memory interface to the private

25

memory bus of processor B **232** and processor C **234**, respectively, of front side bus **220**. Private memory interfaces **250**, **260** each pass the response received to a location in a buffer at the tail of outgoing storage B and

5   C **252**, **262**, respectively. Each outgoing storage **252**, **262** is a conventional storage device, such as memory or disk storage, arranged as a conventional double buffer, with a head and a tail as described above with respect to FIFOs **286**, **288**. As will be described below, it is not necessary

10   to utilize separate outgoing storages **252**, **262** as a single outgoing storage can be used, with each private memory interface **250**, **260** writing into the single outgoing storage **252** or **262**, although it can simplify the circuitry to have individual outgoing storages **252**, **262** to avoid contention

15   issues. The responses are written in blocks having a fixed or variable length as described above. In one embodiment, each processor **234**, **236** writes to a single memory location, and the respective private memory interface **250**, **260** manages placing it into a buffer in the respective outgoing

20   storage **252**, **262** and updating the respective tail pointer. In another embodiment, each processor **232**, **234** reads the tail in the respective outgoing storage **252**, **262** and stores the response or other communication into the private memory buffer location in the respective outgoing storage **252**, **262**

indicated by the tail pointer in the respective outgoing storage **252, 262,** and updates the tail pointer.

Outgoing storages **252, 262** are monitored by one or more outgoing interface manager **254, 264.** Each outgoing

5 interface manager **254, 264** checks the head and tail pointer for the outgoing storage **252, 262** it is monitoring, and if they are different, it takes the block indicated by the tail pointer in outgoing storage **252, 262** in a manner similar to that described above for FIFOs **286, 288,**

10 provides the block to an outgoing communication interface **256** or **266** described in more detail below, and updates the tail pointer in the respective outgoing storage **252, 262** to point to the next block in that outgoing storage **252, 262.**

Outgoing communication interfaces **256, 266** provide at

15 their respective outputs **258, 268** a communication corresponding to each block it receives. Outgoing communications interfaces **256, 266** provide such in a proper protocol and encapsulation, such as TCP/IP messages for the Internet coupled to outputs **258, 268** or TCP/IP messages in

20 Ethernet frames to an Ethernet coupled to outputs **258, 268.**

The blocks are thus provided/output "directly" and "sineinterruptusly", meaning, the I/O buses of the computer system with which the system **200** serves are not used nor do

27

interrupts to the system processors result from the output

of the communications containing the blocks. Again,

conventional 2 port or multi port RAMs as described above

may be used for outgoing storage **252, 262**, and in such

5   embodiment, the blocks are provided "sinebusly", without

the use of the main memory buses as described above.

It is not necessary for each outgoing interface

manager **254, 264** to take blocks only from its respective

outgoing storage **252, 262** as either outgoing interface

10   manager **254, 264** may take blocks from either outgoing

storage **252, 262**. Furthermore, it is not necessary for a

one to one relationship of outgoing communication

interfaces **256, 266** to outgoing interface managers **254, 264**

as any relationship of any number to any number may be

15   used. Furthermore, a common set of outgoing storage may be

used in place of outgoing storages **252** and **262**. Dashed

lines in the Figure are indicated to show connections in

alternative embodiments.

E.   Removal of Entries From Dispatch Storage.

20   In one embodiment, entries are removed from the table

in dispatch storage **288**. In one embodiment, each entry is

timestamped and a removal process is run periodically to

remove an entry based on the timestamp as described in more

28

detail below. The periodic removal process may be run by

the processor that performs the routing function, such as

processor A **230**, a different processor **232, 234** or by

incoming interface manager **212**.

5   Each entry may be timestamped in incoming/dispatch

storage **214** before, during or after the time that any

assignment is made by processor A **230** or incoming interface

manager **212** using that entry, or when the entry is added to

the table, as described above. In another embodiment, the

10   timestamping is performed by the processor **232, 234**

processing the block before, during or after such

processing. The timestamp may be the current time and/or

date or an expiration time and/or date in the future, and

the periodic removal process periodically deletes entries

15   older than a certain amount past the timestamp or any

entries past the expiration time. In one embodiment, the

expiration time is closer to the current time if the block

contains an indication that it is the end of a set of one

or more communications, such as a TCP packet with a FIN

20   flag set, and the expiration time is farther from the

current time otherwise. If the timestamp indicates the

current time, the entry may be marked with an indicator

that such an "end of a set of one or more communications"

indication was received, and processor A **230** removes such

29

entries having a timestamp older than the current time less a period which is shorter than those not having such an indication. Preassigned entries in dispatch storage described above need not be timestamped or removed.

5 **E. Method.**

Referring now to Figure **3**, a method of processing input/output of a computer system is shown according to one embodiment of the present invention. A communication is received **310**. Some or all of the communication is stored 10 **312** "directly" in memory "sineinterruptusly", as those terms are defined above.

If there are multiple processors in the computer system, steps **314-320** are performed, and otherwise, they may be skipped. At step **314** the memory is checked to 15 identify whether any communications have been received but not assigned to a processor as described above and an attempt is made to identify a processor corresponding to the communication stored in step **312**. This attempt may include attempting to locate an entry in a table of entries 20 for which a portion or all of the entry matches a portion or all of the communication stored.

If a corresponding processor is not identified **316**, a processor is selected **318** as described above and a

30

correspondence between the selected processor and the stored communication may be recorded **320**, for example by generating an entry into a table as described above. The entry may be timestamped **322** as described above and

5 assigned to the processor identified either in step **314** or in step **320**.

Storage such as shared memory is checked for a communication and if a communication is waiting to be processed, it is retrieved **330** from the storage. The

10 communication is processed **332** as described above. If a response or other communication is to be generated as a result of the processing step **332 334**, the response or other communication is provided **336**, for example by providing it into storage such as private memory and the

15 method continues at step **340**, and otherwise **334**, the method continues at step **340**.

At step **340**, storage such as private memory is checked to identify if any communications have been written as described above, and if so, they are retrieved **340** and

20 provided "directly" from private memory and "sineinterruptusly", and optionally "sinebusly", as those terms are defined above. Shared memory may be used in

place of private memory in steps **336-342** in another

embodiment of the present invention.

Each set of steps **310-312, 314-322, 330-336** and **340-**

342 may be implemented as separately running processes that

5   repeat among each set of steps as indicated by the dashed

arrows in the Figure.

Although the description above assigns communications

to one of multiple processors, in one embodiment, the

communications are assigned to one of several processes

10   running on the same or different processors. In other

embodiments, communications may be assigned to entities

other than processors and processes.